Software Engineering Internship

July 2024

Titanium Forest

Agenda

- Monday
  - Welcome & Orientation
  - Software Engineering & Version Control
  - Understanding APIs
  - Equity Evaluator project
- Tuesday
  - Javascript & React
  - Directus CMS Intro
- Wednesday
  - SQL Databases
  - Using Directus
  - Linux Basics
- Thursday
  - Kanban

Welcome!

infrastructureSquad

infrastructureSquad is an initiative that promotes IT Infrastructure Operations as a vehicle to teach youth and encore career seekers about the practical applications of computer science and engineering. We provide hybrid training, mentorship, and opportunities for participants to work on real-world products and services that support their local communities.

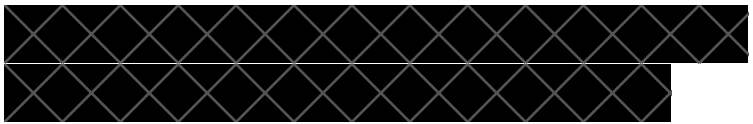
Titanium Forest

Internship Goals

- Introduce Software Engineering
- Provide real-world experience with tools of the trade
- Kick-start the “Equity Evaluator” app
- Have fun!

Orientation: Important Stuff

- Official working hours: 0900-1700, M-F
- Dress code: you must wear clothes!
- Code of conduct
  - Don’t be a jerk
  - Support others
  - Be science-based
  - Have a good time!
- Contact me



Orientation: Other Stuff

- Folder: Notebook & some Wizard Zines (wizardzines.com)
- Desktops: Linux with PopOS 22.04 distro
- Home page: interns.infrastructuresquad.com
- Pick a workstation & login
- Introduction to Linux videos
  - Linux in 100 seconds
  - 100+ Linux Things You Need to Know

Software Engineering

If you know how to code, are you a software engineer?

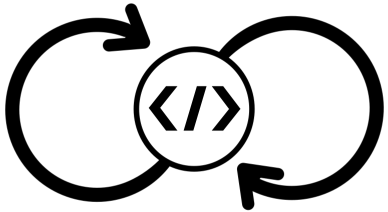
Software Engineering in a Nutshell

Software Engineering is the application of engineering principles to software development in a systematic method

Why?

- Ensures reliability and quality of software
- Facilitates project management and maintenance
- Enables scalability and efficiency in software solutions
- Reduces costs and time-to-market for software products

Software Development Life Cycle (SDLC)

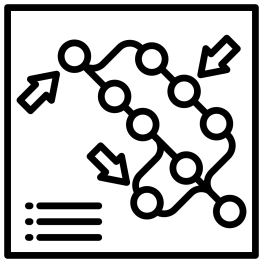


- Planning:
  - Define project scope & objectives
  - Gather requirements & create specifications
  - Conduct research and/or feasibility studies
- Development:
  - Writing and compiling code
  - Designing software architecture
  - Implementing algorithms and data structures
- Testing:
  - Verifying software functionality
  - Identifying and fixing bugs
  - Ensuring software meets requirements
- Deployment:
  - Releasing software to production
  - Managing versions and updates
  - Monitoring performance in the real world
- Maintenance:
  - Updating software to adapt to new requirements
  - Fixing bugs post-deployment
  - Improving performance and usability

Version Control

Why is version control important?

Why Version Control?



- Facilitates collaboration
  - Multiple developers can work on the same project simultaneously
  - Manage changes from different folks
- Provides backup & restore capabilities
  - Keeps history of all changes
  - Allows recovery in case of bugs or data loss
- Promotes accountability and auditing
  - Detailed log of who did what and when
  - Provides an archaeological record: “Why did we do that?!”
- Branching & merging
  - Supports creating branches for new features & bug fixes
  - Enables merging changes into main branch seamlessly

Git Basics

- Git: a distributed version control system by Linus Torvalds in 2005
- Distributed Architecture
  - Each developer has a local copy of the entire repository
  - Facilitates offline work and distributed collaboration
- Fast Performance
  - Optimized for speed and efficiency
  - Handles large projects with ease
- Branching and Merging:
  - Supports lightweight branching and efficient merging
  - Encourages experimentation and feature development
- Staging Area:
  - Allows fine-grained control over what gets committed
  - Facilitates incremental changes and testing

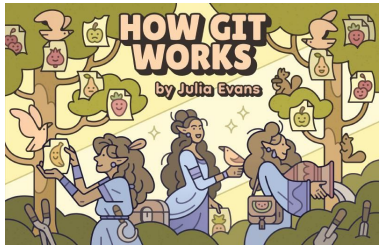
Git Platforms

- Provide repository hosting + extra features
- Public
  - GitHub
  - GitLab
  - Bitbucket
- Self-hosted
  - Gitea (we'll be using this)




Git-ting Started

- Who are you?
  - `git config --global user.name "Your Name"`
  - `git config --global user.email "your.email@example.com"`
- Create a new repo
  - `git init reponame`
- Add new file or unstaged changes
  - `git add mycoolthing.js`
- What's going on?
  - `git status`
- What's changed
  - `git diff`
- Commit changes
  - `git commit`



Git Branching

- Create a new branch & switch to it
  - `git switch -c mynewthing`
- See what branches exist
  - `git branch`
- Switch to another branch that exists
  - `git switch otherbranch`
- Delete branch
  - `git branch -d dead-to-me`
  - `git branch -D really-go-away`
- What happened?
  - `git log`

Git Collaboration

- Clone remote repo
  - `git clone {url}`
- Add remote to local repo
  - `git remote add origin {url}`
- Fetch changes from remote
  - `git fetch`
- Fetch changes **and** merge into current branch
  - `git pull`
- Push changes
  - `git push`

Git Collaboration Problems

- Merge conflicts

```
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
```

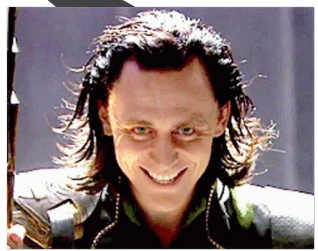
- Stale local branch

Pull Requests & Code Reviews

- Pull request (PR): request to merge changes to a tree
- Provide commentary in the PR as to what your changes do
- Once submitted, it’s ready for review
- One or more people (or programs) will review your request
- The approvers will approve or reject the request
- If approved, you should merge your change
- Reviewers ensure code quality and consistency
- PR process encourages collaboration and knowledge transfer (bus insurance)

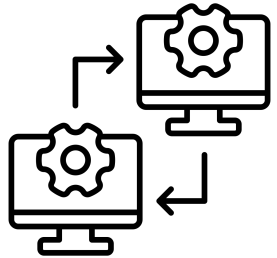
Let's Practice!

- Clone the repo
- Create a branch
- Push your branch
- Create a PR
- Review a PR (feel free try out rejection!)
- Merge your PR if it was approved
- Switch back to 'main' branch & pull



Understanding APIs

What is an API?



An API (Application Programming Interface) is a set of rules and protocols that allows one software application to interact with another

- API models
  - Libraries
  - Remote Procedure Call (RPC)
  - REST
  - SOAP
  - GraphQL
  - gRPC
- Data encoding formats
  - JSON
  - XML
  - Protobuf
  - x-www-form-urlencoded

JSON vs XML


```
{"users":[{"id":1,"name":"John
Doe","email":"john.doe@example.com","posts
":[{"id":101,"title":"GraphQL
Introduction","content":"An introduction
to
GraphQL...","timestamp":"2024-07-05T12:34:
56Z"},{"id":102,"title":"Advanced
GraphQL","content":"Deep dive into
GraphQL...","timestamp":"2024-07-06T14:20:
00Z"}]},{"id":2,"name":"Jane
Smith","email":"jane.smith@example.com","p
osts":[{"id":103,"title":"REST API vs
GraphQL","content":"Comparing REST and
GraphQL...","timestamp":"2024-07-07T09:15:
45Z"}]}]}
```



```
  <users>
    <user id="1" name="John Doe"
email="john.doe@example.com">
      <posts>
        <post id="101" title="GraphQL Introduction"
content="An introduction to GraphQL..."
timestamp="2024-07-05T12:34:56Z"/>
        <post id="102" title="Advanced GraphQL"
content="Deep dive into GraphQL..."
timestamp="2024-07-06T14:20:00Z"/>
      </posts>
    </user>
    <user id="2" name="Jane Smith"
email="jane.smith@example.com">
      <posts>
        <post id="103" title="REST API vs GraphQL"
content="Comparing REST and GraphQL..."
timestamp="2024-07-07T09:15:45Z"/>
      </posts>
    </user>
  </users>
```

REST APIs

- REST = Representational State Transfer (Fielding, 2000)
- Resource-based, identified by URIs
- Use HTTP methods for specific purpose: GET, POST (create), PUT (replace), PATCH (modify)
- Each request is stateless (independency; idempotency)
- Resources represented in a specific format (e.g. JSON, XML)
- Uniform interface (typically HTTP)
- Infrastructure layers between client and server OK

Play Around with REST APIs

- Use Insomnia or curl
- https://apipheny.io/free-api/
- https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/
- https://github.com/public-apis/public-apis